Midterm 2 - Section 001 (A)

Name:

Penn State access ID (xyz1234) in the following box:

Student ID number (9XXXXXXXX):

Instructions:

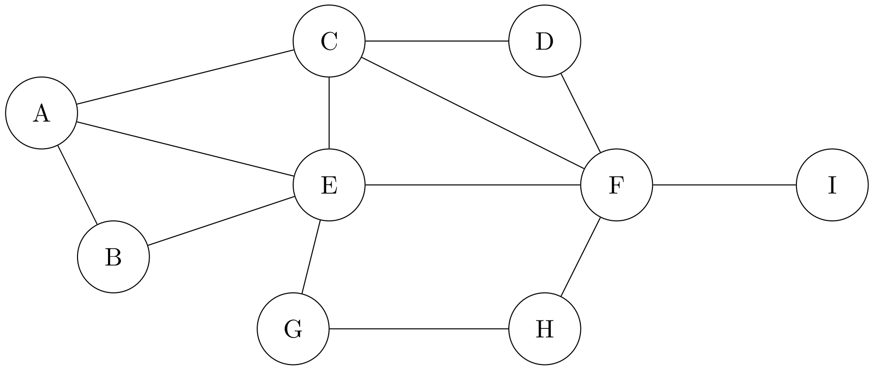
- Answer all questions. Read them carefully first. Be precise and concise. Handwriting needs to be neat. Box numerical final answers.
- Please clearly write your name and your PSU access ID (i.e., xyz1234) in the box on top of **every page**.
- Write in only the space provided. You may use the back of the pages only as scratch paper. **Do not write your solutions on the back of pages!**
- **Do not write outside of the black bounding box**: the scanner will not be able to read anything outside of this box.
- We are providing one extra page at the end if you need extra space. Make sure you mention in the space provided that you answer continues there.

Good luck!

Name:	**PSU Access ID (xyz1234):**

Graphs *(30 points)*

1. Perform depth-first search (DFS) on the following graph; whenever there is a choice of vertices, pick the one that is alphabetically first. Find the pre and post visit numbers for each vertex.



Pre-visit numbers:

(a) The pre-visit number of node A is:

[✓ 1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

(b) The pre-visit number of node B is:

1 [✓ 2] 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

(c) The pre-visit number of node C is:

1 2 3 [✓ 4] 5 6 7 8 9 10 11 12 13 14 15 16 17 18

(d) The pre-visit number of node D is:

1 2 3 4 [✓ 5] 6 7 8 9 10 11 12 13 14 15 16 17 18

(e) The pre-visit number of node E is:

1 2 [✓ 3] 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

(f) The pre-visit number of node F is:

1 2 3 4 5 [✓ 6] 7 8 9 10 11 12 13 14 15 16 17 18

(g) The pre-visit number of node G is:

1 2 3 4 5 6 7 [✓ 8] 9 10 11 12 13 14 15 16 17 18

(h) The pre-visit number of node H is:

1 2 3 4 5 6 [✓ 7] 8 9 10 11 12 13 14 15 16 17 18

(i) The pre-visit number of node I is:

1 2 3 4 5 6 7 8 9 10 [✓ 11] 12 13 14 15 16 17 18

Name:	**PSU Access ID (xyz1234):**

Post-visit numbers:

(j) The post-visit number of node A is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 [✓ 18]

(k) The post-visit number of node B is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 [✓ 17] ◯18

(l) The post-visit number of node C is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 [✓ 15] ◯16 ◯17 ◯18

(m) The post-visit number of node D is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 [✓ 14] ◯15 ◯16 ◯17 ◯18

(n) The post-visit number of node E is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 [✓ 16] ◯17 ◯18

(o) The post-visit number of node F is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 [✓ 13] ◯14 ◯15 ◯16 ◯17 ◯18

(p) The post-visit number of node G is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 [✓ 9] ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(q) The post-visit number of node H is:

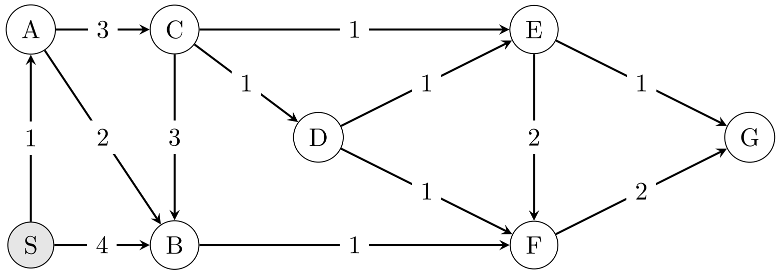
◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 [✓ 10] ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

(r) The post-visit number of node I is:

◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 [✓ 12] ◯13 ◯14 ◯15 ◯16 ◯17 ◯18

Name:	**PSU Access ID (xyz1234):**

2. Run Djikstra's Priority Queue Algorithm for three iterations starting with the source node S, and provide the shortest path distances obtained in the below graph.



(a) Distance to S at end of 3rd iteration:

[✓ 0] ◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯∞

(b) Distance to A at end of 3rd iteration:

◯0 [✓ 1] ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯∞

(c) Distance to B at end of 3rd iteration:

◯0 ◯1 ◯2 [✓ 3] ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯∞

(d) Distance to C at end of 3rd iteration:

◯0 ◯1 ◯2 ◯3 [✓ 4] ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 ◯∞

(e) Distance to D at end of 3rd iteration:

◯0 ◯1 ◯2 ◯3 ◯4 [✓ 5] ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 [✓ ∞]

(f) Distance to E at end of 3rd iteration:

◯0 ◯1 ◯2 ◯3 ◯4 [✓ 5] ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 [✓ ∞]

(g) Distance to F at end of 3rd iteration:

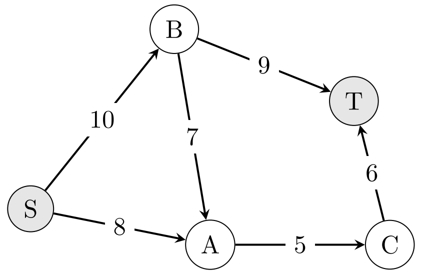
◯0 ◯1 ◯2 ◯3 [✓ 4] ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 [✓ ∞]

(h) Distance to G at end of 3rd iteration:

◯0 ◯1 ◯2 ◯3 ◯4 ◯5 ◯6 ◯7 ◯8 ◯9 ◯10 ◯11 ◯12 ◯13 ◯14 ◯15 ◯16 ◯17 ◯18 [✓ ∞]

Name:	**PSU Access ID (xyz1234):**

3. Given the flow network depicted below with source node S and sink node T, fill in the values for the residual capacities after Ford-Fulkerson terminates (both forward and backward capacities for each pair of vertices even if it is zero). What is the maximum amount of flow that the network would allow to flow from source to sink node?
Note that how you find, and in what order, (S, T)-augmenting paths does not matter.



(a) Edge (S, B):
[✓ 0] [✓ 1] ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10

(b) Edge (B, S):
○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 [✓ 9] [✓ 10]

(c) Edge (S, A):
○0 ○1 ○2 [✓ 3] [✓ 4] ○5 ○6 ○7 ○8 ○9 ○10

(d) Edge (A, S):
○0 ○1 ○2 ○3 [✓ 4] [✓ 5] ○6 ○7 ○8 ○9 ○10

(e) Edge (A, B):
[✓ 0] [✓ 1] ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10

(f) Edge (B, A):
○0 ○1 ○2 ○3 ○4 ○5 [✓ 6] [✓ 7] ○8 ○9 ○10

(g) Edge (T, B):
○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 [✓ 9] ○10

(h) Edge (B, T):
[✓ 0] ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10

(i) Edge (A, C):
[✓ 0] ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10

(j) Edge (C, A):
○0 ○1 ○2 ○3 ○4 [✓ 5] ○6 ○7 ○8 ○9 ○10

(k) Edge (C, T):
○0 [✓ 1] ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10

(l) Edge (T, C):
○0 ○1 ○2 ○3 ○4 [✓ 5] ○6 ○7 ○8 ○9 ○10

(m) The value of the maximum flow is:
○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 ○11 ○12 ○13 [✓ 14] ○15 ○16 ○17 ○18

Name: **PSU Access ID (xyz1234):**

True/False *(20 points)*

True or false? Fill in the correct bubble. No justification is needed.

	T	F	
1.	○	●	In a DFS on a directed graph $G = (V, E)$, if u was explored before v, and there is a path from u to v, then $post(v)$ is greater than $post(u)$.
2.	○	●	There is no known polynomial-time algorithm to solve the maximum flow problem.
3.	○	●	If the weights of all edges in a weighted graph are unique, then there is always a unique shortest path from a source vertex to a destination vertex.
4.	○	●	The longest simple path (a path of distinct vertices) in a graph can be found by multiplying the cost of each edge by -1 and then running the Bellman-Ford algorithm.
5.	○	●	Multiplying each edge weight in a graph by a constant k will not affect the shortest path of a graph.
6.	○	●	If an iteration of the Ford-Fulkerson algorithm on a network places flow $f(e) > 0$ through an edge $e = (u, v)$, then in every later iteration, the flow through (u, v) is at least $f(e)$.
7.	●	○	In a weighted graph $G = (V, E)$ where all edge weights are integers and at most 1000, we can find the shortest paths from a given vertex s to any other vertex in $O(\|V\| + \|E\|)$ time.
8.	●	○	Given any flow network, the Ford-Fulkerson algorithm will always find the max flow for any order of choosing $s - t$ paths.
9.	○	●	The meta-nodes formed by each Strongly Connected Component (SCC) output by the SCC algorithm are output in topological order (i.e., the first outputted meta-node should be furthest to the left in the ordering).
10.	●	○	Every directed acyclic graph (DAG) has at least one topological ordering.

Name:	**PSU Access ID:**

Short Questions. *(20 points)*

1. Given a graph $G = (V, E)$ with possibly negative weights but no negative cycle, design an efficient algorithm to compute the shortest path from every node in the graph that ends in a certain node t. Describe your algorithm in a high level and establish its running time.

   Solution: Reverse the directions of all edges to get a new graph G^R. Then the question is equivalent to finding the shortest path from node t to all other nodes in G^R. Since there could be negative weights but no negative cycle, we can apply Bellman-Ford algorithm to find shortest paths from ndoe t. Running time for reversing the graph is $O(|V| + |E|)$. Running time for Bellman-Ford algorithm is $O(|V| \cdot |E|)$. Overall, the running time is $O(|V| \cdot |E|)$.

2. Consider a directed graph $G = (V, E)$ with nonnegative edge weights (lengths) and a starting vertex s. Define the bottleneck of a path to be the maximum length of one of its edges. Show how to modify Dijkstra's algorithm to compute, for each vertex $v \in V$, the smallest bottleneck out of the bottlenecks of all shortest $s - v$ paths. Your algorithm should run in $O((m + n) \log n)$ time using priority queues, where $m = |E|$ and $n = |V|$ denote the number of edges and vertices, respectively.

   Solution: Use a tuple (i, w_i) to indicate the value of the bottleneck along the path to node i. Each time when we update the shortest distance to a node i, we will also update w_i. Let the previous node of i be j. Then we set w_i to $\max\{w_j, C(j, i)\}$. If the distance to node i is the same via nodes j and k, we will pick the node with smaller bottleneck, i.e. $\min\{\max\{w_j, C(j, i)\}, max\{w_k, C(k, i)\}\}$. Since we just add one more step (i.e., updating bottleneck value and picking smaller one) during the distance update procedure, the running time should be the same as Dijkstra's algorithm, which is $O((m + n) \log n)$.

Name:	**PSU Access ID:**

3. Suppose some one presents you with a solution to a max-flow problem on some flow network. Give a linear time algorithm to determine whether the solution does indeed give a maximum flow.

   Solution: This is the same question in worksheet 8 problem 2.

   First, we verify that the given flow is a valid flow on the network. It is valid if it satisfies the capacity constraint and the flow conservation constraint. Assuming it is a valid flow, note that for every max-flow, there is no $s - t$ path in the residual graph. Therefore, to determine if the given flow is a max-flow, we search for an $s - t$ path in the residual graph.

   Checking the flow conservation condition can be done in $O(|V| + |E|)$ time by iterating over every edge and storing one incoming and one outgoing sum for each node. At edge (u, v), increment node u's outgoing variable and v's incoming variable. After the sums have been computed, iterate over every node and check that the sums are equal. Checking the capacity condition takes $O(|E|)$ time, simply iterate over every edge. Constructing G^f takes $O(|V| + |E|)$ time. Running BFS on G^f takes $O(|V|+|E|)$ time since G^f has $|V|$ vertices and $\leq 2|E|$ edges. Therefore, the algorithm is $O(|V|+|E|)$, which is linear.

4. Recall that a forward and tree edge $e = (u, v)$ both satisfy the condition pre[u] $<$ pre[v] $<$ post[v] $<$ post [u]. Give a method to distinguish whether an edge $e = (u, v)$ is a forward or tree edge using only the pre and post numberings of vertices (i.e., without knowledge of any other edge.)

   Solution: If an edge e = (u, v) is a forward edge, it should have another vertex x satisfies two conditions: pre[u] ¡ pre[x] ¡ post[x] ¡ post[u] and pre[x] ¡ pre[v] ¡ post[v] ¡ post[x]. First condition means that u is the ancestor of x, and second condition means that x is the ancestor of v. Otherwise it should be a tree edge.

Name:	**PSU Access ID (xyz1234):**